Università degli Studi di Parma
Dipartimento di Ingegneria e Architettura
Sistemi operativi e in tempo reale – a.a. 2022/23

Esercizi di programmazione concorrente
Soluzione mediante Monitor

prof. Stefano Caselli
stefano.caselli@unipr.it

Il problema del ponte

- Ponte a senso unico alternato, due direzioni (Sx e Dx)

- Sul ponte possono trovarsi al più MAX veicoli, ovviamente tutti nella medesima direzione
- Il ponte è una risorsa da utilizzare in modo efficiente

Il problema del ponte: Soluzione con Regione Critica Condizionale

```
#define N 3     // MAX veicoli sul ponte
var bridge  shared record
                    direction: (EAST, WEST) initial EAST;
                    car_count: integer initial 0;
     end
procedure  enter_bridge (my_dir);
  begin
   region bridge when (car_count=0 or ((car_count<N) and (direction=my_dir))
          do begin
                    car_count++ ;
                    direction := my_dir;
          end
  end
```

Il problema del ponte: Soluzione con Regione Critica Condizionale

```
procedure  leave_bridge;
 begin
   region bridge when 1 do car_count-- ;
 end
```

P1

…

enter_bridge (EAST);

<transit bridge>

leave_bridge;

P2

…

enter_bridge (WEST);

<transit bridge>

leave_bridge;

Problema del ponte

- Soluzione con *Monitor* ed *una sola variabile condizione*

```
type  bridge = monitor {
  car_count:     integer;
  dir:           (E,W);
  queue:         condition;
  procedure entry enter_bridge(my_dir);
   begin
        if (car_count = MAX) or ((car_count > 0) and (direction <> my_dir))
        then queue.wait;
        car_count++ ;
        dir := my_dir;
   end
```

Problema del ponte

```
procedure entry leave_bridge;
 begin
        car_count-- ;
        queue.signal;
  end
begin car_count = 0; end
}   //end monitor
```

Problema del ponte

- La soluzione 1 si propone di usare una sola variabile condizione, tuttavia ...
- La `queue.signal` in `leave_bridge` potrebbe risvegliare un veicolo nella direzione opposta! → Occorre testare la condizione di sincronizzazione all'interno di un *while* anzichè entro un *if*
- Anche sostituendo *while* ad *if:*
  - Se il ponte era pieno e il primo thread in attesa rappresenta un veicolo in direzione opposta, nessuno prende il posto del veicolo segnalante → sottoutilizzazione della risorsa
  - Quando cambia la direzione va in esecuzione un solo thread, con sottoutilizzazione del ponte e possibile scavalcamento dei thread in attesa da parte di nuovi arrivati
- Con una sola variabile condizione occorre effettuare una `cond.broadcast` al cambio di direzione di attraversamento

Problema del ponte

- Soluzione 2: variabili condizione associate staticamente alle due direzioni di transito; previene anche starvation
- Di seguito elementi principali della soluzione: manca sintassi completa del tipo di dato astratto/monitor

```
#define MAX 3                          // massimo numero auto sul ponte
#define MAX_TRANSIT  100     // num transiti prima di inversione direzione
#define DX  0    // destra
#define SX  1     // sinistra
int nmacc=0;                                    // numero auto sul ponte
int dir;                                        // direzione auto sul ponte (DX o SX)
int in_coda_a_sx=0, in_coda_a_dx=0;      // num auto in coda
int transitate;              // num auto transitate da ultima inversione di direz.
condition ponte_libero_per_sx, ponte_libero_per_dx;
```

Problema del ponte

```
procedure entry enter_sx() {          // eseguita da veicolo che entra da SX
 if ((nmacc>0 && dir==DX) || (nmacc==MAX) || (in_coda_a_sx>0)) {
   in_coda_a_sx++;
   ponte_libero_per_sx.wait;
   in_coda_a_sx--;
   };
 nmacc++;
 if (nmacc==1)  {
   dir=SX;
   transitate=0;
   };
};
```

Problema del ponte

```
procedure entry leave_sx() {          // veicolo entrato da SX esce dal ponte
 nmacc--;
 if (in_coda_a_dx>0)
    transitate++;
 if (nmacc==0)
    for(i=0; i<MAX; i++)
       ponte_libero_per_dx.signal;
 else if ((in_coda_a_sx>0) && (transitate<MAX_TRANSIT))
     ponte_libero_per_sx.signal;
};
```

- *procedure entry* enter_dx() e *procedure entry* leave_dx(): simmetriche alle altre, con dx e sx scambiati

Problema del ponte

- Cosa succede se MAX=1, nella enter_sx() ?
- Meglio identificare il cambio di direzione con variabile di stato specifica

- In leave_sx() mediante un *ciclo for* si fa il risveglio di MAX veicoli nella direzione opposta, al cambio di direzione di attraversamento del ponte
- E' una soluzione corretta, ma sarebbe possibile realizzare invece un *risveglio a catena* dei veicoli?

Problema del ponte

- Soluzione 3: usa due variabili condizione non associate staticamente alle direzioni di percorrenza del ponte

```
type ponte = monitor {
  var    dir_corrente: (da_EST, da_OVEST);
         numero_auto:  0..MAX;
         dir_sbagliata, ponte_pieno: condition;
  procedure entry enter_bridge(dir: (da_EST, da_OVEST));
   begin
         if (dir_corrente<>dir) and (numero_auto>0) then dir_sbagliata.wait;
         if numero_auto = MAX then ponte_pieno.wait;
         dir_corrente := dir;                          // ok to enter
         numero_auto := numero_auto + 1;
   end
```

Problema del ponte

```
procedure entry leave_bridge();
  begin
        numero_auto := numero_auto - 1;
        ponte_pieno.signal;
        if numero_auto = 0 then dir_sbagliata.signal;
  end
begin            // inizializzazione stato istanze
   numero_auto :=0; dir_corrente := da_EST;
end
}       // fine def tipo monitor
```

- Esempio d'uso:

```
var fornovo: ponte;

fornovo.enter_bridge(da_EST);
transit_bridge;
fornovo.leave_bridge;
```

Problema del ponte

- La soluzione precedente *non è efficiente*, perchè al cambio di direzione solo uno dei veicoli in direzione opposta impegna il ponte

```
procedure entry enter_bridge(dir: (da_EST, da_OVEST));
    begin
        if (dir_corrente<>dir) and (numero_auto>0)
        then begin
                dir_sbagliata.wait;
                dir_sbagliata.signal;          // risveglio a catena !!
        end
        if numero_auto = MAX then ponte_pieno.wait;
        dir_corrente := dir;                   // ok to enter
        numero_auto := numero_auto + 1;
    end
```

Problema del ponte

```
procedure entry leave_bridge();
  begin
        numero_auto := numero_auto - 1;
        if numero_auto > 0 then ponte_pieno.signal; // solo se ci sono auto
        else dir_sbagliata.signal;
  end
```

Problema del ponte

- Soluzioni per avere efficienza in fase di commutazione della direzione:
  a) broadcast (possibile elevato num. di context switch, accettabile se per un numero max predefinito e ragionevole di thread in attesa)
  b) ciclo for di cond.signal (idem, accettabile se esiste un numero max predefinito e ragionevole di thread in attesa) – Esempi:

```
for(i=0; i<Num_Auto_in_Attesa_dx; i++)
    ponte_libero_per_dx.signal;                          // NO!!
for(i=0; i<MAX_Auto_sul_Ponte; i++)
   ponte_libero_per_dx.signal;                           // SI'
for(i=0; i<min(MAX_Auto_sul_Ponte,Num_Auto_in_Attesa_dx); i++)
   ponte_libero_per_dx.signal;                           // MEGLIO
```

Problema del ponte

- Soluzioni per avere efficienza in fase di commutazione della direzione:
  c) risveglio a catena al lato dei thread in attesa

- Per un ponte su strada di montagna con passaggi sporadici: vanno bene tutte le soluzioni ed anche il broadcast!

- Le soluzioni ai problemi di concorrenza diventano critiche per l'efficienza in presenza di thread con interazioni strette e frequenti

Deposito sci

- L'albergo di una remota località turistica invernale mette a disposizione dei suoi clienti un locale contenente attrezzi per la pratica di sci alpino, sci di fondo e snowboard. Tutti i clienti richiedono calzature della stessa misura ed attrezzi della medesima lunghezza.
  - Per lo sci alpino i clienti richiedono: sci da discesa (SA), racchette (R) e scarponi (S);
  - Per lo sci di fondo richiedono: sci da fondo (SF) e racchette (R), mentre utilizzano scarpette personali;
  - Per lo snowboard richiedono: scarponi (S) e tavola (T).
- Di ciascun tipo di attrezzatura il deposito dispone di una quantità Ni (con i appartenente al set {SA, R, S, SD, T}), ampia ma in generale insufficiente a soddisfare tutte le possibili richieste dei clienti.
- Per favorire la pratica sportiva ai clienti viene inoltre richiesto di praticare due sport diversi nell'arco della giornata.

Deposito sci

- Si risolva il problema di programmazione concorrente utilizzando il costrutto monitor, indicandone in modo esplicito la semantica. Si preveda, oltre alla definizione del tipo, la istanziazione della variabile e la traccia di esecuzione dei thread. Non è consentito l'impiego delle primitive broadcast e queue sulle variabili condizione.
- La soluzione proposta deve essere esente da problemi di attese inutili o attive, deadlock e starvation.

- Nota: leggere e comprendere la consegna!

Deposito sci - soluzione /1/

- Semantica MESA

```
#define ALPINO 0
#define FONDO 1
#define SNOWBOARD 2

type  deposito = monitor;
{
   int            NSA, NR, NS, NSF, NT;
   condition      alpino_coda, fondo_coda, snowboard_coda;
   int            alpino_coda_dim, fondo_coda_dim, snowboard_coda_dim;
```

Deposito sci - soluzione /2/

```
procedure entry sport_start (int sport) {
   switch (sport) {
   case ALPINO:   /* condizione unica entro while */
                  /* per ritestare la condizione dopo il risveglio */
         while ((NSA == 0) || (NR == 0) || (NS == 0)) {
                  alpino_coda_dim ++;                  /* Si mette in coda */
                  alpino_coda.wait ();
                  alpino_coda_dim --;
         }
         NSA --;              /* Prende attrezzi e va */
         NR --;
         NS --;
         break;
```

Deposito sci - soluzione /3/

```
case FONDO:
        while ((NSF == 0) || (NR == 0)) {      /* while con condizione unica */
                fondo_coda_dim ++;             /* Si mette in coda */
                fondo_coda.wait ();
                fondo_coda_dim --;
        }
                /* Prende attrezzi e va */
        NSF --;
        NR --;
        break;
```

Deposito sci - soluzione /4/

```
case SNOWBOARD:
        while ((NS == 0) || (NT == 0)) {        /* while con condizione unica */
                /* Si mette in coda */
                snowboard_coda_dim ++;
                snowboard_coda.wait ();
                snowboard_coda_dim --;
        }
        NS --;                  /* Prende attrezzi e va */
        NT --;
        break;
} /* end switch */

}  /* end of entry */
```

Deposito sci - soluzione /5/

```
procedure entry  sport_end (int sport) {
   switch (sport) {
   case ALPINO:
          NSA++;
          NR++;
          NS++;
          if ((fondo_coda_dim > 0) && (NSF > 0) {    // Segnala a sportivo in att.
                  /* C'e' almeno una racchetta (appena consegnata),  */
                  /* pertanto se ci sono sci di fondo puo' essere opportuno  */
                  /* segnalarlo ai fondisti in attesa */
                  fondo_coda.signal ();
                  }
```

Deposito sci - soluzione /6/

```
if ((snowboard_coda_dim > 0) && (NT > 0) {
        /* Stessa cosa per snowboard */
        /* Priorita' a snow e fondo: con una restituzione */
        /*  e' possibile liberazione di due clienti */
        snowboard_coda.signal ();
}

if ((alpino_coda_dim > 0)) {
        alpino_coda.signal ();
}
break;
```

Deposito sci - soluzione /7/

- Note:
- Dopo il risveglio, i thread provvedono da soli a verificare le proprie condizioni; sarebbe quindi possibile eseguire delle signal incondizionate
- In realtà, se si risvegliano e si bloccano nuovamente con la wait, essi passano dalla prima all'ultima posizione della fila, il che è poco *fair* (ammesso che la coda venga servita in modo FIFO dal supporto che realizza il Monitor)
- Verificare le condizioni prima di risvegliarli riduce il problema, mentre per eliminarlo occorre un test più stretto dello stato

Deposito sci - soluzione /8/

```
case FONDO:
        NSF++;
        NR++;
        /* Segnala a quelli in attesa */
        if ((alpino_coda_dim > 0) && (NSA > 0) && (NS > 0)) {
                alpino_coda.signal ();
        }
        if (fondo_coda_dim > 0) {
                fondo_coda.signal ();
        }

        break;
```

Deposito sci - soluzione /9/

```
  case SNOWBOARD:
        NS ++;
        NT ++;
        /* Segnala a quelli in attesa */
        if ((alpino_coda_dim > 0) && (NSA > 0) && (NR > 0)) {
              alpino_coda.signal ();
        }
        if (snowboard_coda_dim > 0) {
              snowboard_coda.signal ();
        }
        break;
} /* switch */
```

Deposito sci - soluzione /10/

```
/* Procedura di inizializzazione, eseguita automaticamente */
 {
        NSA = NSA_INIZIALE;
        NR = NR_INIZIALE;
        NS = NS_INIZIALE;
        NSF = NSF_INIZIALE;
        NT = NT_INIZIALE;
        alpino_coda_dim = 0;
        fondo_coda_dim = 0;
        snowboard_coda_dim = 0;
 }

}       /* end definizione del monitor */
```

Deposito sci

- creazione istanza:

  deposito ski-rent;

- traccia di esecuzione dei thread (generico thread sportivo):

  ...
  ski-rent.sport_start (ALPINO);
  <scia>
  ski-rent.sport_end (ALPINO);
  <pausa pranzo>
  ski-rent.sport_start (FONDO);
  <scia>
  ski-rent.sport_end (FONDO);
  ...

Problema dei fumatori incalliti (Patil, 1971)

- Tre fumatori incalliti si trovano in una stanza con un venditore di articoli per il fumo. Per fumare una sigaretta ogni fumatore necessita di tre ingredienti: tabacco, carta e fiammiferi, dei quali il venditore ha ampia disponibilità. Un fumatore dispone di tabacco personale, un altro di propria carta, e l'ultimo di propri fiammiferi
- L'azione inizia quando il venditore pone due degli ingredienti sul tavolo, per consentire ad un fumatore di commettere un atto insalubre. Quando un fumatore ha finito risveglia il venditore, che deposita altri due ingredienti (a caso) sbloccando in tal modo un fumatore
- Si scrivano programmi di controllo per i fumatori ed il venditore, dapprima sincronizzandone le attività mediante una regione critica condizionale e successivamente mediante un monitor